WEEK 5

Q- Search given Key in BST

```
 class Solution {
public:
  TreeNode* searchBST(TreeNode* root, int val) {
      if(root==NULL)
      {
         return NULL;
      }
      queue<TreeNode*> que;
      que.push(root);
      while(!que.empty())
      {
         TreeNode* temp=que.front();
         que.pop();
         if(temp->val==val)
         {
            return temp;
            break;
         }
         if(temp->left)
         {
            que.push(temp->left);
         }
         if(temp->right)
         {
            que.push(temp->right);
         }
```

```
        }
        return NULL;
    }
};
```

Q- Construct BST from given keys

```
class Solution {
public:
    TreeNode* makeTree(vector<int> nums,int s,int e)
    {
        if(s > e)
            return NULL;

        int mid = (s+e)/2;
        TreeNode* root = new TreeNode(nums[mid]);

        root->left = makeTree(nums,s,mid-1);

        root->right = makeTree(nums,mid+1,e);

        return root;

    }

    TreeNode* sortedArrayToBST(vector<int>& nums) {

        return makeTree(nums,0,nums.size()-1);

    }
```

```
};
```

Q- Construct Binary Search Tree from Preorder Traversal

```
 class Solution {
    int i=1;
public:
    TreeNode* bstFromPreorder(vector<int>& preorder) {
        int size=preorder.size();
        if(size==0)
      return NULL;
   TreeNode *b=new TreeNode(preorder[0]);
   TreeNode *h=b;
   int k;
   for(int i=1;i<size;i++){
      h=b;
      k=0;
      while(k==0){
      if(h->val>preorder[i]){
         if(h->left==NULL){
            TreeNode *p=new TreeNode(preorder[i]);
            h->left=p;
            k++;
         }
         else{
            h=h->left;
         }
      }
      else{
         if(h->right==NULL){
            TreeNode *p=new TreeNode(preorder[i]);
```

```
            h->right=p;
            k++;
          }
          else{
            h=h->right;
          }
        }
        }
    }

    return b;
     }
};
```

Q- Validate Binary Tree is BST

```
class Solution {
public:
    bool isBST(TreeNode* root, TreeNode* min, TreeNode* max){
        if(root==NULL){
            return true;
        }

        if(min!=NULL && root->val <= min->val){
            return false;
        }
        if(max!=NULL && root->val >= max->val){
            return false;
        }
```

```
        bool leftValid = isBST(root->left, min, root);
        bool rightValid = isBST(root->right, root, max);

        return leftValid and rightValid;
    }

    bool isValidBST(TreeNode* root) {
        return isBST(root, NULL, NULL);
    }
};
```

Q- Find LCA of two nodes in Binary Search Tree

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root->val>p->val &&root->val>q->val)return lowestCommonAncestor(root->left,p,q);

            if(root->val<p->val &&root->val<q->val)return lowestCommonAncestor(root->right,p,q);

            return root;
    }
};
```

Q- Find the successor of a given key in BST

```
void find_p_s(Node* root,int a,
            Node** p, Node** q)
{
    if(!root)
        return ;
```

```
    find_p_s(root->left, a, p, q);

    if(root&&root->data > a)
    {
        if((!*q) || (*q) && (*q)->data > root->data)
                *q = root;
    }

    else if(root && root->data < a)
    {
        *p = root;
    }

    find_p_s(root->right, a, p, q);
}
```

Q-Floor in BST

```
int floorInBST(TreeNode<int> * root, int x)
{
       int ceil=-1;
   while(root){
      if(root->val==x){
         ceil=root->val;
         return ceil;
      }

      if(x<root->val){
         root=root->left;
```

```
      }
      else{
         ceil=root->val;
         root=root->right;
      }
   }
   return ceil;
}
```

Q- Ceil in a BST

```
int findCeil(BinaryTreeNode<int> *root, int key){
   int ceil=-1;
   while (root){
      if (root->data==key){
         ceil = root-> data;
         return ceil ;

      }
      if ( root ->data>key){
         ceil= root->data;
         root = root->left;
      }
      else{
         root= root->right;
      }
   }
   return ceil;
}
```

Q- Find K-th Largest element in BST

```
node* kthlargest(node* root,int& k)
{
        if(root==NULL)
        return NULL;

        node* right=kthlargest(root->right,k);
        if(right!=NULL)
        return right;
        k--;

        if(k==0)
        return root;

        return kthlargest(root->left,k);
}
```

Q- Find K-th Smallest element in BST

```
node* kthsmallest(node* root,int &k)
{
        if(root==NULL)
        return NULL;

        node* left=kthsmallest(root->left,k);
        if(left!=NULL)
        return left;
        k--;
        if(k==0)
        return root;
```

```
            return kthsmallest(root->right,k);
}
```

Q- Find a Pair with a given sum in BST

```
class Solution {
public:
    unordered_map<int,int>mp;
   bool findTarget(TreeNode* root, int target) {
       if(root==NULL){return false;}

       if( mp.find(target-(root->val)) != mp.end())
       {return true;}

       mp[root->val]++;

       return (findTarget(root->left,target)||findTarget(root->right,target));

   }
};
```

Q- BST iterator

```
class BSTIterator {
public:


    stack<TreeNode* > st;

   void inorder(TreeNode* root){
       if(root==NULL) return;
```

```
        st.push(root);
        inorder(root->left);
    }

    BSTIterator(TreeNode* root) {
        inorder(root);
    }

    int next() {
        TreeNode* temp=st.top();
        st.pop();
        inorder(temp->right);
        return temp->val;
    }

    bool hasNext() {
        return !st.empty();
    }
};
```

Q- Maximum sum bst in binary tree

```
class Solution {
public:
    int ans=0;
    vector<int> dfs(TreeNode* root){
        if(!root)
            return {0,INT_MAX,INT_MIN,true};
        vector<int>left=dfs(root->left);
        vector<int>right=dfs(root->right);
```

```
        if(left[3] && right[3] && left[2]<root->val && right[1]>root->val){
            int sum=left[0]+right[0]+root->val;
            ans=max(sum,ans);
            return {sum,left[1]==INT_MAX?root->val:left[1],right[2]==INT_MIN?root->val:right[2],true};
        }
        return {0,INT_MIN,INT_MAX,false};
    }
    int maxSumBST(TreeNode* root) {
        dfs(root);
        return ans;
    }
};
```

Q- Serialize and Deserialize binary tree

```
string serialize(TreeNode* root) {
        if(!root) return "";

        string s ="";
        queue<TreeNode*> q;
        q.push(root);
        while(!q.empty()) {
          TreeNode* curNode = q.front();
          q.pop();
          if(curNode==NULL) s.append("#,");
          else s.append(to_string(curNode->val)+',');
          if(curNode != NULL){
              q.push(curNode->left);
              q.push(curNode->right);
```

```
        }
    }
    return s;
}

TreeNode* deserialize(string data) {
    if(data.size() == 0) return NULL;
    stringstream s(data);
    string str;
    getline(s, str, ',');
    TreeNode *root = new TreeNode(stoi(str));
    queue<TreeNode*> q;
    q.push(root);
    while(!q.empty()) {

        TreeNode *node = q.front();
        q.pop();

        getline(s, str, ',');
        if(str == "#") {
            node->left = NULL;
        }
        else {
            TreeNode* leftNode = new TreeNode(stoi(str));
            node->left = leftNode;
            q.push(leftNode);
        }

        getline(s, str, ',');
```

```
            if(str == "#") {
                node->right = NULL;
            }
            else {
                TreeNode* rightNode = new TreeNode(stoi(str));
                node->right = rightNode;
                q.push(rightNode);
            }
        }
        return root;
    }
```

Q- Flatten Binary Tree to Linked List

```
class Solution {
public:
    TreeNode* prev= NULL;

    void flatten(TreeNode* root) {
        if(root==NULL) return;

        flatten(root->right);
        flatten(root->left);

        root->right=prev;
        root->left= NULL;
        prev=root;
    }
};
```

Q- Find Median from Data Stream

```
class MedianFinder {
public:
    priority_queue<int> maxHeap;
    priority_queue<int, vector<int>, greater<int>> minHeap;
    int size;
    MedianFinder() {
        size = 0;
    }

    void addNum(int num) {
        size++;
        int lSize = maxHeap.size();
        int rSize = minHeap.size();
        if (lSize == 0) {
            maxHeap.push(num);
        } else if (lSize == rSize) {
            if (num < minHeap.top())
                maxHeap.push(num);
            else {
                int temp = minHeap.top();
                minHeap.pop();
                minHeap.push(num);
                maxHeap.push(temp);
            }
        } else {
            if (num > maxHeap.top())
                minHeap.push(num);
            else {
```

```
            int temp = maxHeap.top();
            maxHeap.pop();
            maxHeap.push(num);
            minHeap.push(temp);
        }
      }
    }

    double findMedian() {
      if (size % 2) return maxHeap.top();
      return (maxHeap.top() + minHeap.top())/double(2);
    }
};
```

Q- Kth largest stream in Stream

```
class KthLargest {
public:
    int a;
    multiset<int> ms;
    KthLargest(int k, vector<int>& nums)
    {
        a=k;
        for(int i=0;i<nums.size();i++) ms.insert(nums[i]);
    }

    int add(int val)
    {
        ms.insert(val);
        int count=1;
```

```
        for(auto it:ms)
        {
            if(count==ms.size()-a+1) return it;
            else count++;
        }
        return 0;
    }
};
```